Aerodynamic Simulation on Massively Parallel Systems

JOCHEM HÄUSER AND HORST D. SIMON

EUROPEAN SPACE RESEARCH AND TECHNOLOGY CENTER
EUROPEAN SPACE AGENCY
2200 AG NOORDWIJK, THE NETHERLANDS
AND
COMPUTER SCIENCES CORPORATION
NAS APPLIED RESEARCH BRANCH
NASA AMES RESEARCH CENTER
MOFFETT FIELD, CA 94035, USA

Abstract

This paper briefly addresses the computational requirements for the analysis of complete configurations of aircraft and spacecraft currently under design to be used for advanced transportation in commercial applications as well as in space flight. The discussion clearly shows that massively parallel systems are the only alternative which is both cost effective and on the other hand can provide the necessary TeraFlops, needed to satisfy the narrow design margins of modern vehicles. It is assumed that the solution of the governing physical equations, i.e., the Navier-Stokes equations which may be complemented by chemistry and turbulence models, is done on multiblock grids. This technique is situated between the fully structured approach of classical boundary fitted grids and the fully unstructured tetrahedra grids. A fully structured grid best represents the flow physiscs, while the unstructured grid gives best geometrical flexiblity. The multiblock grid employed is structured within a block, but completely unstructured on the block level. While a completely unstructured grid is not straightforward to parallelize, the above mentioned multiblock grid is inherently parallel, in particular for MIMD machines. In this paper guidelines are provided for setting up or modifying an existing sequential code so that a direct parallelization on a massively parallel system is possible. Results are presented for three parallel systems, namely the Intel hypercube, the Ncube hypercube, and the FPS 500 system. Some preliminary results for an 8K CM2 machine will also be mentioned. The code run is the 2D grid generation module of **Grid∗** , which is a general 2D and 3D grid generation code for complex geometries. A system of nonlinear Poisson equations is solved. This code is also a good testcase for complex fluid dynamics codes, since the same datastructures are used. All systems provided good speedups, but message passing MIMD systems seem to be best suited for large multiblock applications.

1 Parallel Computing for Aerospace Applications

At present several new spaceplanes as well as aircraft are under design in Europe, the United States and Japan. The aerodynamics and aerothermodynamics behavior of these vehicles has to be accurately kown to make them actually fly. Since the time of Prandtl, windtunnels played a prominent role in the design and analysis of configurations. Since the last two decades CFD (Computational Fluid Dynamics) has made substantial progress and can now be considered as an alternative to windtunnel measurements for certain types of flow, e.g. inviscid flow for complex 3D geometries both for perfect gas and real gas effects or can be used to supplement windtunnel data. In the support of the Hermes space plane, which may serve as an example here, numerous 3D Euler calculations have been performed in ESTEC using different algorithms and grids, achieving a very good agreement of predicted pitching moment coefficients for Ma numbers between 10 and 25 and angles of attack up to 30 degrees. Similar computations have been performed for the Space Shuttle [6]. However, prediction of turbulence level and transition are a major problem for computations and windtunnels alike. Usage of windtunnel data for free flight is difficult since extrapolation leads to additional uncertainties. For high speed flow when chemistry plays a role, a scaled down model can not be subjected to the same flow phenomena as the real vehicle. Reynolds number, temperature and species disscociation cannot be doubled at the same time in the windtunnel. Also, the amount of information obtained from experiments is very limited. On the other hand, grid generation for complex geometries is still a major issue , especially when very different length scales have to be resolved.

Flexibility and versatility of numerical simulation allow the separation of physical effects and to investigate the influence of geometry. In addtion, new visualization software, e.g. Visual3 from MIT [4], have implemented all the standard windtunnel visualization techniques so that a very realistic picture of the flow can be obtained. Although turn around time of a moderately large simulation run is relativley small when compared to windtunnel testing, parametric studies, for example of heat flux with a Navier-Stokes code for a complete vehicle for a full flight trajectory is not feasible because of excessive computing time. A grid size of 10 million points for viscous calculations for a 3D vehicle has been estimated for accurate heat flux studies. Needing about 8,000 iterations with an implicit N-S solver to reach the steady state for a Mach number of 25 with an operational cost of approximately 8,000 Flops per point and iteration, a total of $6.4 * 10^{14}$ floating point operations has to be performed. For equilibrium real gas calculations, the additional cost is between $20\% - 30\%$. Here a laminar case was assumed. Even with the Teraflop machine available , some 650 seconds would be needed. Moreover, since 1 Teraflop sustained has to be delivered, the announcement of a 10 Teraflop peak machine should be awaited. Inclusion of complex turbulence models and nonequlibrium flow will substantially increase the computing time. Since aerodynamics and aerothermodynamics have a direct impact on the structure, a realistic evaluation of a vehicle eventually has to couple the flow solution with an aeroelasticity code. In some cases, a coupling of aerodynamics with Maxwell's equations may be needed to obtain vehicle shape optimization.

These enormous computational requirements can not be satiesfied with the convential von Neumann architecture. First, the speed of light is a limiting factor in signal propagation, demanding miniaturization to an extent not feasible since the dissipation of generated heat demands a certain chip size. Second, this approach will not be cost effective in comparison with of the shelf chips. Hence, instead of using a single or a small number of extremely powerful processors, a large set of reasonably powerful from the shelf processors (Intel i860, Mips R4000 etc.) can be assembled, each with its own private memory and then made to communicate via the exchange of messages. This architecture is called MIMD (Multiple Instructions Multiple Data). This massively parallel architecture is particularly well suited to multiblock problems, where one or more blocks are mapped on a single processor.

The above remarks should not lead to the conclusion that windtunnel testing will be obsolete in the near future. Experimental testing is mandatory to confirm the results computed and to validate codes as well as to provide information where computation is not feasible or not accurate enough, e.g. in cases of severe flow separation. However, CFD and in particular parallel CFD may have a more prominent role in aerospace design in the coming years.

2 Parallel Programming Languages for CFD

This section is based on the first author's work parallelizing **Grid*** and **NSS*** and therefore expresses mostly his view. First, it is believed that a new parallel language for CFD is not needed. The currently existing languages Fortran, C, and C++, enhanced by parallel constructs, will be able to provide the necessary functionality. Fortran has been the traditional language in the scientific and engineering field since the late 50's, and a large number of codes is available. Before the three languages are briefly discussed, a list of desirable features of a general purpose language is given below.

- desirable features of a high level parallel language for CFD :
    1. wide availability
    2. portability (both for sequential and parallel machines)
    3. maintaining and debugging of code
    4. rapid prototyping code (high productivity)
    5. compact code
    6. vesatile loop and conditional statements
    7. dynamic storage allocation
    8. recursion

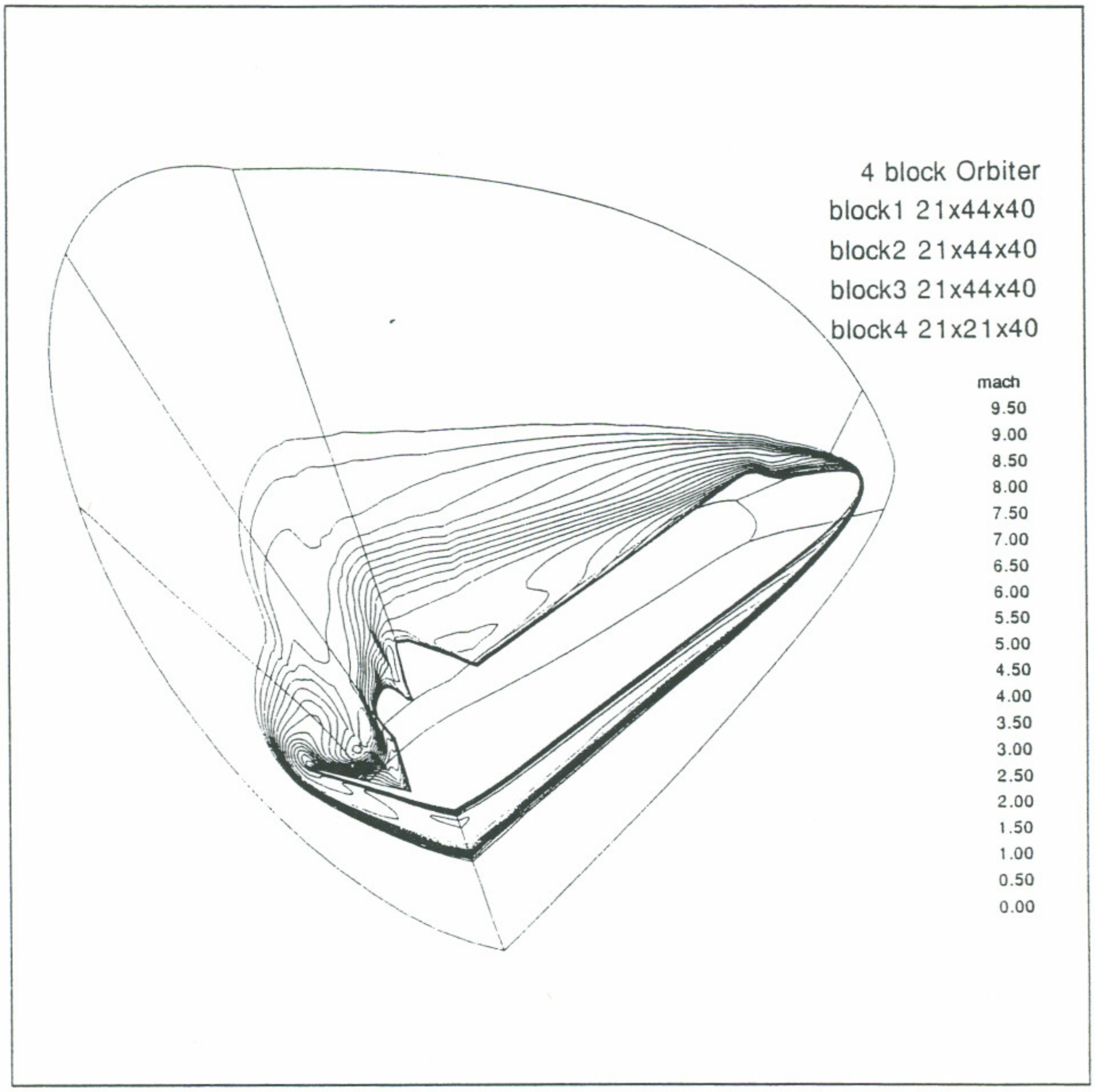

Figure 1: Mach number distribution for the Shuttle orbiter obtained from the N-S solver ESA-NSS using a 4 block grid. Free stream values are $Ma = 25$. A perfect gas model was used since the calculations simulate wind tunnel conditions of R3, Chalais.

9. safety (handling of internal error situations as well as wrong input data)
10. pointers
11. advanced data types (structures)
12. classes (information hiding)
13. operator overloading (simple example : + for addition of both real numbers and matrices)
14. conditional compilation (implementation of conditional statements to run the same code on different parallel machines)

It is believed that the aspects of software engineering in the solution of fluid dynamics and theromodynamics problems have not been given due attention by the engineering community. The prevailing attitude of "putting it on the computer " has lead in many cases to inadequate but costly "solutions" (the word tool is avoided here). Many of the topics of the above list have not been considered in software projects, leading to poorely debugged and unsafe code and also to large cost overruns of the project. Since the parallelization of a code adds to its complexity, the following remarks are worthwhile to consider. The widely used F77 does not possess any modern constructs. The proposed upgrade F90 will have some modern features available. The widely used ANSI C has all the features listed, except classes. Dynamic storage allocation is somewhat limited, since arrays have to be dynamically allocated via pointers. All features listed are provided by C++. The following recommendation, motivated below, is therefore given.

- C++ is the language of choice for the 90's, but not all of its advanced features will be needed for CFD.

As has been aforesaid most of todays compute intensive codes are written in Fortran. **Grid*** and **NSS*** both were started in Fortran. Soon it became obvious that the implementation of advanced data structures was impossible in F77 (the same holds for F90) and one either would have to write a preprocessor or to find a more suitable tool. For example, manipulation of a complete block was impossible, because a data structure named "block" could not be defined. A block consists of faces, a face is built from edges, an edge is formed by points, and eventually a point comprises a set of 3 floating point numbers. In addition, the size of an object is not known a priori. It is not known how many blocks comprise the SD (solution domain) or how many points are on a face. The number of objects is determined at run time only. Hence, the most natural way is to allocate memory for an object when it is needed and to free it when this is no longer the case. If this cannot be done, resources are wasted. The 2D Fortran version of **NSS*** needs 2.5 times more memory than the C version. On the IBM Risc 6000/540 model the execution time of the Fortran code (highest optimization level) is 10% higher than for the C version and its code size is approximately three times larger. In order to achieve some type of functionality in Fortran, all kinds of artificial constructions have to be made. This quickly leads to a messy code, which can be maintained by the author only. Moreover, nearly all of these codes are unsafe for the user, that means, a new set of input parameters often leads to an undefined program state. A substantial reduction in productivity is the result where the user has to debug the source code for months. Many of these problems can be avoided by using the proper tools, namely ANSI C and C++. For example, **Grid*** with all its modules [11] comprises only 3,000 lines of code. It has also been experienced that modifications of the code are straightforward since objects are modified, but the relations between objects remain the same, increasing code safety. The additional advantage of C++ is that classes can be defined which contain both data structures and functions and then operations among classes can be performed. This leads to well defined interfaces between classes, and code modifications have only local effects. Operator overloading allows to define any type of operation between classes. A rudimentary installation will also be available in F90. It is therefore believed that using ANSI C or C++ will be much more cost effective both in terms of human and of hardware resources. For the much more sophisticated programming tasks lying ahead, code safety will be extremely important. Also for that reason ANSI C and C++ should be preferred.

The major difficulty in using C is the new way of thinking that is needed. Thinking in objects, advanced data structures, and pointers as well as in classes is new for the Fortran programmer and therefore may take some time.

3 Multiblock Codes and MIMD Machines

Grid* [11] is a general multiblock code, used to generate overlapping, multiblock grids for arbitrary 3D geometries. The output of this code is directly coupled to the 3D Navier-Stokes solver **NSS*** , which produces a solution without any further user interaction, provided that the grid has the quality needed for the flow physics to be simulated. The asterisk in the code names stands for the wildcard symbol, indicating that they are a collection of C routines, based on the UNIX toolbox concept. Both codes have the same data structures, and therefore the same communication behavior. Of course, the ratio between communication and computation for the two codes is different. The N-S solver needs a factor of 100 more floating point operations per grid point than the grid generator. Any efficieny numbers obtained from the parallelization of the grid generator can be considered to be substantially exceeded for the flow solver. It should be noted that the multiblock concept is introduced because of the complex geometry. That is, even on a sequential computer communication between blocks is necessary. Hence, multiblock codes lead to an inherently parallel program code. In order to determine the blocksize for **NSS*** for which a high speedup can be expected, the following information has to be known.

- assumption : a grid of about 1 million grid points is distributed on 100 processors
- assumption : the floating point performance for the i860 is 5 MFlops

- the blocksize is approximately 30*30*10 internal points
- the solution scheme of **NSS$*$** needs an overlap of 2 points at each face (3rd order accuracy in shock free area)
- the memory overhead caused by multiblock is therefore about 66 %
- the number of IPs (internal points) is 9000 and the number of BPs (boundary points) is 6000 per block
- the compute time for **NSS$*$** is 7200 ms /iteration/block (based on 5 MFlops)
- the communication time for **NSS$*$** is 360 ms /iteration/block based on a sustained transfer rate of 1MByte/sec.
- for interblock communication a maximum of 6 send and 6 receive messages per block is needed; in total 12 faces have to be sent :
- the amount of communication in bytes per block and per iteration is: $\approx$ 360 KB = 5 variables $(\rho, \rho u, \rho v, \rho w, e)\ of$ $8bytes * 4faces * 30 * 30 + 8faces * 30 * 10 +$ $5\ variables\ (\delta\rho, \delta\rho u, \delta\rho v, \delta\rho w, \delta e) of$ $8bytes * 2faces * 30 * 30 + 4faces * 30 * 10$
- communication time : $t_{send} = 6 * 9 * 10^{-2} + 3.6 * 10^5 * 1 * 10^{-3} ms \ \approx 360\ ms$ where the first term nenotes the latency. A communication speed of 1MB/s has been assumed [5].

For **NSS$*$** the values of the 5 conservative variables and their increments have to be sent across each face of a block. Since the increments denote differences, an overlap of 1 is needed only. The communication overhead is determined from the formula $t_{send} = 0.047 t_{calc}$ that is communication overhead for sending messages is $\approx$ 5 % . The time for encoding and decoding messages has to be added leading to an estimated 8% overhead, based on the possibilty of subdividing the grid into blocks of exactly equal size. That this can be achieved for a complete vehicle is demonstrated in [5]. Therefore, even for a N-S solver like **NSS$*$** , communication will be an appreciable part of the overall computation time. The conclusion is that the communication bandwidth used in the iPSC 860 should be upgraded by at least a factor of 10. A machine with the codename Sigma, announced by Intel for the 3rd quarter of 92 will have a peak communication speed of about 200 MB/s, which is approximately an increase of two orders of magnitude over the current iPSC 860. If there was a Teraflop machine consisting of 1000 nodes, each with a compute power of 1 GFlop, a sustained communication rate of 50 MB/s has to be achieved. It seems to be the case that under the communication load of a 3D N-S solver, 5 to 10 % of the peak communication speed can be obtained [5].

4 System Commands and System calls for CFD Multiblock Codes

The Intel prallel system commands shall serve as an example for the parallelization calls needed by an application to allocate resources. In order to achieve the best portability and for the ease of implementation, the following guidelines should be considered already when a sequential code is developed or being modified. These guidelines can be directly implemented in the sequential version to minimize the parallelization effort. Hence, it should be possible to obtain a version of the code that is usable on both sequential and massively parallel systems by isolating interblock communication.

- system commands (UNIX makefile) instead of system calls (user program) should be used to control the parallel machine (getcube, load, relcube etc.). The system commands and the system calls managing the resources have the same functionality, but work on different levels. Separation of resource management (makefile) and user program will enhance portability.
- minimize number of system calls for message passing (csend(),crecv(),irecv() etc.) in the user program.
- all nodes should be on the same level, i.e., there should be no host program. Since all blocks are treated equal and are loosely synchronized by message passing via block faces, a host is not needed.
- use CFS (Concurrent File System) for input/output : for a large number of processors the number of files can be several hundred, since a data file for each block exists.
- subdivide input into global data (all nodes) and local data (single node).
- length in bytes of message buffer should be a multiple of 4 to avoid internal alignment calculations.
- check for pending messages before a function is left to reduce possible deadlocks.

Parallelizing the 2D version of **Grid∗** and working on the parallelization of **NSS∗** , it has been found that only a few system calls are needed, which can be applied in a straightforward manner. The system calls used provide general information and perform the message passing. In the following the functions along with their argument list are explained. In order to achieve a loosely coupled synchronization among the blocks, send and receive commands are used. At the end of each iteration, each block issues 6 receive and 6 send commands (if the face is a physical boundary no information needs to be exchanged). Only after all receive commands have been processed, block boundaries can be updated and after that the next iteration can be started. Therefore, the following sequence of receive and send commands is chosen. First, 6 nonblocking receive commands are issued, irecv(), that is the code does not stop and wait until the receive is completed but continues in its execution. The receive statements are immediately followed by 6 blocking sends, csend(). The send command is only waiting for the availability of a communication channel to the destination processor, but does not wait for an acknowledgement of the receiving node. After that, the code has to wait again until all receives are completed. Function msgwait() using the return value of the corresponding irecv() call performs this task. The reason for this sequence is as follows. If information for a node is available, but no receive has been issued, the information has to be buffered and has to be read a second time when the receive is specified.

A protocol that first synchronizes a block with its neighbors and the issues a receive and a send looping over all neighbors may half the communication time [14] by making use of the full duplex properties of the iPSC links.

- system calls for parallelization of CFD multiblock codes
    1. mynode() : returns node id
    2. time in seconds (double precision) : dclock()
    3. set breakpoint using irecv return value : msgwait(ret_val)
    4. returns msgid of pending or received message : infotype()
    5. nonblocking receive : irecv(msgid,r_buf,length)
    6. blocking receive : crecv(msgid,r_buf,length)
    7. blocking send : csend(msgid,s_buf,length,node,pid)
    8. pending message ? : cprobe(msgid)

In addition, commands for global communication may be needed. The function parameters have the following meaning.

- int msgid,length,node,pid,ret_val
- char *r_buf, *s_buf
- msgid is the message identifier
- r_buf, s_buf are pointers to the names of the receive and send buffers
- length is the message length in bytes
- node denotes the processor number to which message is sent
- pid=1 (only 1 process allowed on the i860)

5 General Parallelization Tasks for CFD Multiblock Codes

5.1 The Four Major Parallelization Tasks for Multiblock CFD Codes

General experience with CFD codes on massively parallel systems is described in [8], [9], and [10]. In the parallelization of a general 3D CFD multiblock code four major tasks have to be achieved. These are the following.

1. parallelization of Input/Output: local
   solution : partition input in local and global data and use CFS, for output each node writes directly via the CFS.

2. parallelization of generation of initial solution: local plus CFS
   solution : see Sec. 5.3. The algorithm is local, but datafiles of neighboring blocks have to be read from disk.

3. parallelization of block to block communication: global
   solution: when a plane (face) is exchanged between neighboring blocks, it is sent into a buffer, rotated (to get the right orientation), and sent to the destination node. This procedure is the same both for the sequential and the parallel codes.

4. parallelization of real gas table lookup for the flow solver: local
   solution: the present size of real gas tables is approximately 2 MB. They are stored in each pocessor. With the real gas table generation technique from Vinokur and Liu a size of less than 512 KB can be guarateed for density and energy in the range of of $10^{-9} < \rho < 10^{3}$ and $10^{-5} < \epsilon < 10^{8}$ where ρ has dimension kg/m^{3} and ϵ is measured in *Joule* [13].

5.2 General Rules : Input/Output Parallelization

Input Data for **Grid**∗ and **NSS**∗ have been split into two categories, namely global and local data. The topology file named "top.cmd" which contains information about block connectivity and block dimensions is global and has to be stored in each node. The size of this file is $n*(3+6*8)*sizeof(int)$ bytes where n denotes the number of nodes, that is for 128 nodes some 74 KB are needed, which is a negligible amount of storage. The geometry files are named "bloc#no" and are local. The current bloc number is denoted by no. This file contains the grid point coordinates. The size is $3*I*J*K*sizeof(float)$

bytes, where I, J, K denote maximum block dimensions in the respective coordinate directions. For the example chosen, the size is $3 * 30 * 30 * 10 * 8 \approx 216$ KB for an 8byte floating point number. The files can be moved to the CFS by the system call cread() or the standard fprintf() can be used, which causes an overhead of 50 %. The output of a node is directly sent to the CFS and assembled via a Shell script if necessary. The structure of the "top.cmd" file for a 3D one block input example is shown below.

```
# connectivity information for 3D example
# bloc number
# bloc dimensions I, J, K
# face numbering: 1=down, 2=front, 3=left, 4=right, 5=back, 6=up
# face type: 0=fixed boundary,
#            1=fixed (interpolate
#            initial solution by starting from that face)
#             2=matching(extending boundary),
#             3=matching(nonextending)
# neighboring_bloc_number (1...n, 0 if no neighbor)
# neighboring_face_number (1...6, 0 if no neighbor)
# neighboring_face_orientation (0...3), rotation 0, π/2 etc.
# CF= Control Function (grid point clustering)
# neighboring_bloc_number (1...n, 0 if no neighbor)
# neighboring_face_number (1...6, 0 if no neighbor)
# neighboring_face-usage (1...8, rotation and mirroring)
#
\cntrl3d
1
6 6 6
1 1 0 0 0 0 0 0
2 1 0 0 0 0 0 0
3 1 0 0 0 0 0 0
4 3 2 3 0 0 0 0
5 3 3 2 0 0 0 0
6 0 0 0 0 0 0 0
⋮
```

The structure of the data file "bloc#no.inp" for a 3D multiblock is of the following form.

```
# input data is a 3d plane
\plane3d
# dimension of plane
6 6
# grid point coordinates : (x,y,z)
0.000000 0.000000 0.000000
1.000000 0.000000 0.000000
```

```
2.000000 0.000000 0.000000
3.000000 0.000000 0.000000
⋮
\plane3d
6 6
0.000000 0.000000 5.000000
1.000000 0.000000 5.000000
2.000000 0.000000 5.000000
3.000000 0.000000 5.000000
⋮
```

This structure works both for the sequential and the parallel case. Reading the data files sequentially does not cause a problem since it is done only once.

5.3 General Rules : Parallelization of Initial Solution Generation

In the following the parallelization of the generation of the initial solution for **Grid**$*$ is shown, which is obtained by interpolation from the boundary point distribution. A good starting solution is essential for the convergence of the elliptic grid generation operators and reduces the number of iterations by an order of magnitude. In general, interpolation is straightforward. However, in the present case interpolation has to be across block boundaries, since the initial solution is determined by interpolating between to corresponding fixed boundaries with an arbitrary number of blocks in between, starting from a fixed side (type 1). Although this seems to be a nonlocal process, it can be made local, since the topology file is stored in each processor and data files of neighboring processors can be read from disk. In order to perform the interpolation, the total distance (number of grid points) between two fixed sides (faces) has to be known. This information is available from the topology file and the coordinates can be read from the data files. Therefore no communication is needed to construct the initial solution in each block. In Fig.2 a 12 block grid for an airfoil is shown along with the initialization procedure. The same algorithm is executed in each node.

[Initialize] all block coordinates to 0, set flag fixed to 0 for all faces in block.
[Loop] over all faces tracing back rays (a ray is a coordinate line in the third direction) passing through all neighboring blocks until a fixed face is encountered.
[Store] respective blockno (topology file) l_bloc, and faceno l_face. Set fixed=1 and calculate distance l_dist (number of gridpoints in the corresponding direction) to fixed face.
[Initialization face ?] if (fixed == 1) {
start from face with no $7 - l\_face$ (opposite face) and trace back rays rays through all neighboring blocks, until a fixed face in encountered.}
[Store] respective blockno, r_bloc and faceno r_face. Calculate distance r_dist.

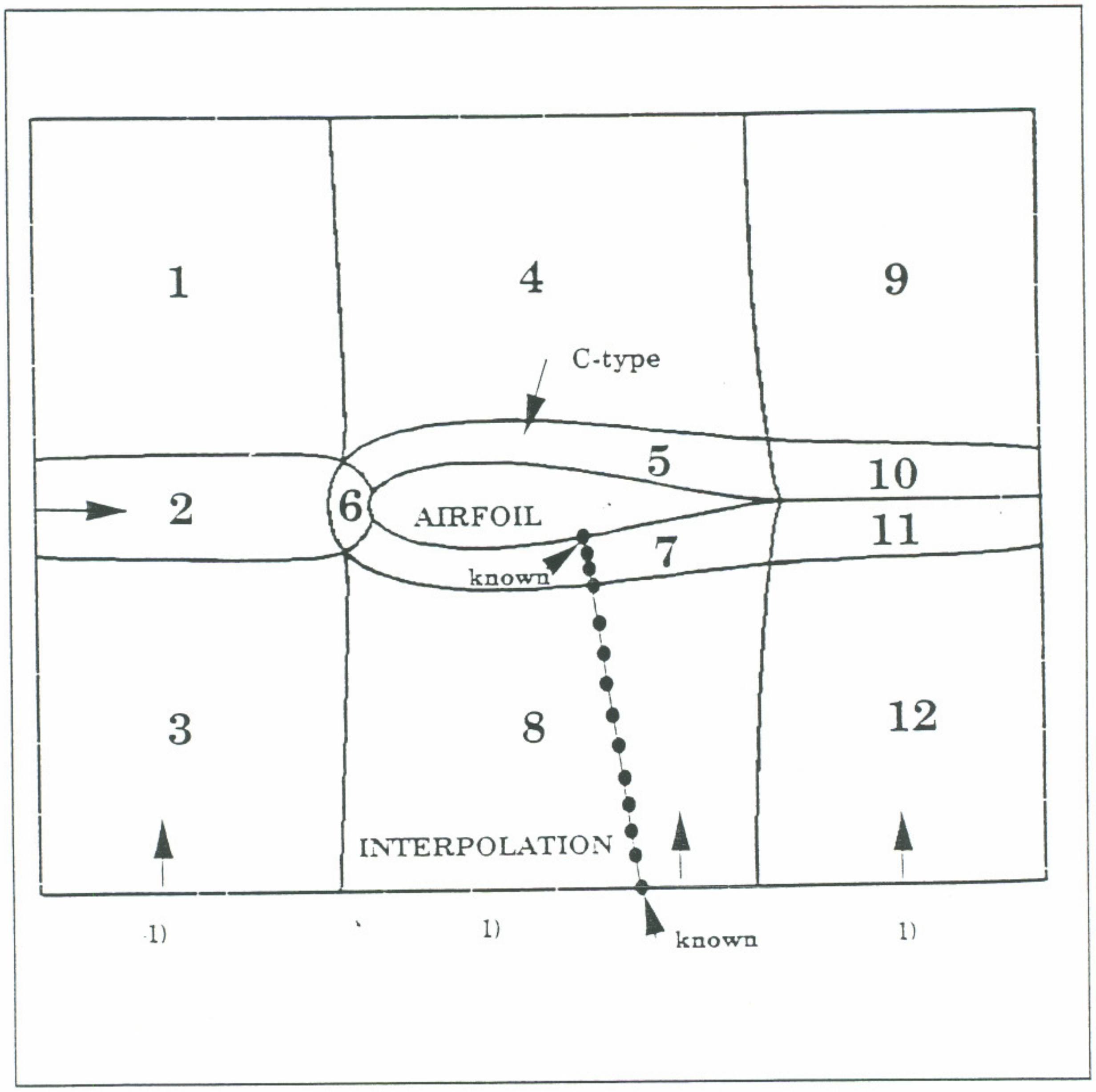

Figure 2: Algebraic Grid Construction: Lines emanating from fixed surfaces are followed through all blocks until another fixed surface is met.

[Read] l_face geometry data from file "bloc#l_face.inp".
[Read] r_face geometry data from file "bloc#r_face.inp".
[Rotate] l_face plane to match local CS of current block.
[Rotate] r_face plane to match local CS of current block.
[Calculate] local distance for interpolation :
dist = l_dist+r_dist+m_dist, where m_dist is the length of current block in the respective coordinate direction.

5.4 General Rules : Block to Block Communications

Interblock communication is performed in the following way. There exists one global CS (coordinate system) in physical space : (x,y,z). To provide the needed geometrical flexibility, each block has its own local CS (I,J,K) in computational space. Any of the 8 orientations (Fig.3), which are theoretically possible between local CSs can be chosen. For each block a function named "savpla" writes overlap faces (type plane3D) into respective buffers. For each receiving block a function named "operate3d" rotates/mirrors each 3D plane to have the right orientation with regard to the neighboring CS. After that function "getpla" gets neighboring plane from the buffer to update the corresponding overlapping face. The following calling sequence of receive and send is considered to be advantageous. It avoids message copying from system to application buffers, which takes place if message arrives before crecv was issued.

```
# request new face information by issuing a nonblocking receive
botid=irecv(1,face_rbuf,sizeof(face_rbuf)
⋮
topid=irecv(6,face_rbuf,sizeof(face_rbuf)
# send out new face information using blocking send
csend(1,face_sbuf,sizeof(face_sbuf)
⋮
csend(6,face_sbuf,sizeof(face_sbuf)
# set breakpoint for nonblocking receives
msgwait(topid)
⋮
msgwait(botid)
```

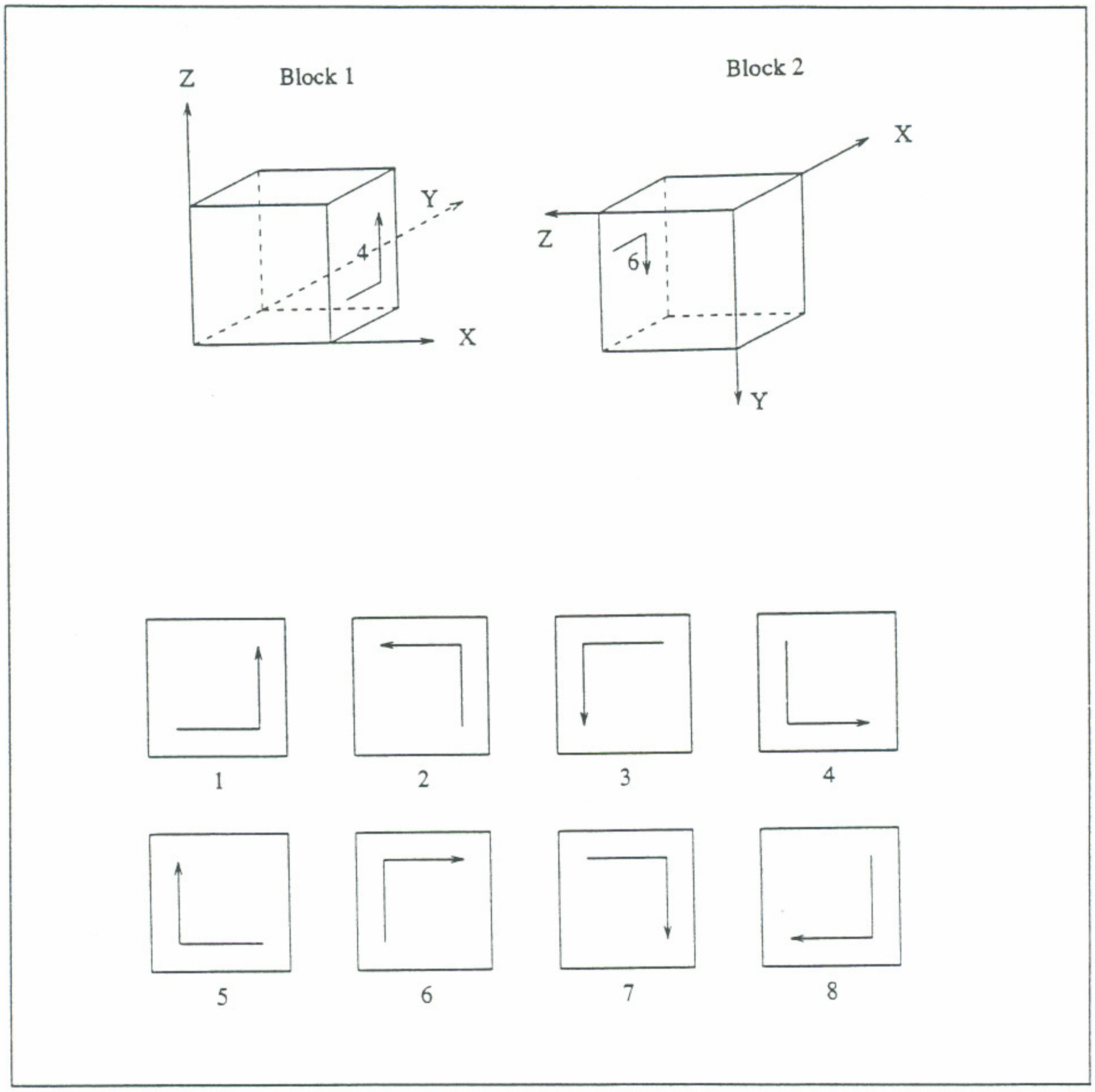

Figure 3: The 8 possible orientations of neighboring faces are shown. The first 4 cases denote rotation, while cases 5 to 8 are obtained by mirroring.

6 Results for the 2D Module of Grid∗ on FPS, Intel, and Ncube Architectures

In the following results for **Grid∗** [11] are presented on several MIMD machines. Similar results can be expected for all codes which are based on partial differential equations using multiblock domains. The SD is subdivided in overlapping blocks with slope continuous grid lines, and therefore no special treatment of block boundaries is needed.

The 2D module of **Grid∗** has been used as a testcase on 3 systems, namely the FPS 500, the Intel iPSC2, and the Ncube system. A simplified version has been also installed on a 8K CM2 from Thinking Machines, but more work is needed to optimize the communication. The goal of the simulations was not a direct comparison of run times, but to find out about the effort needed to parallelize the code and to learn about the usefulness of these machines for this type of application. A system of two nonlinear Poisson equations is solved by SOR, generating a 2D multiblock grid. The code generates grids for any arbitrary geometry and thus is a realistic example for the complex domains encountered in aerospace applications. Based on this experience, a new testcase simulating the bevavior of a 3D multiblock Navier-Stokes code has been developed, but was run so far only on the Intel Gamma and Delta machines at Caltech, using up to 512 processors [1].

The FPS 500 has a shared memory, while Intel and Ncube have a distributed memory. The CM2 from Thinking Machines is a SIMD (Single Instruction Multiple Data) architecture. Since the architecture of the FPS 500 is not widely known, a few explanations will be given. The FPS 500 has one scalar Sparc processor and a parallel matrix procrressor, MP, which uses Intel's i860 chip as the processing element. The MP contains up to 84 i860 chips connected by a crossbar of $8 * 160$ MB/s (peak). When 84 processors are used, 7 buses connecting each 12 processing elements are employed. A 64 MB memory, the so called matrix register, was used. It also contains a vector processor which was not used for this testcase.

All vendors have announced much more powerful systems for the second half of 92. The Intel iPSC2 is a second generation parallel machine, and Intel has announced already its Sigma machine, which is the 4 th generation. Approximately three orders of magnitude separate these two machines, both for the communication behavior as well as for the compute power. Tables 1 to 3 show performances for the Ncube, Intel and FPS systems. No attempt was made to compare the results. First, the hardware is very different, and second, the level of fine tuning for the code versions is very different. In particular, the code for FPS was optimized to minimize cache faults [2]. Since the data cache of the i860 is only 8KB, it is not sure whether this procedure would be successful in 3D, when planes instead of lines have to be handled. Intel and Ncube code are almost compatible. FPS as a shared memory system nedds a different parallelization approach. The extension to several hundreds or thousands of processors for the FPS remains to be demonstrated.

Ncube System						
points per proc	number of processors					
	1	2	4	8	16	32
30×30	1	0.82	0.59	0.36	0.21	0.11
60×30	1	0.88	0.72	0.50	0.32	0.19
60×60	1	0.94	0.84	0.66	0.44	0.31
120×60	1	0.97	0.90	0.81	0.65	0.46
130×120	1	0.98	0.96	0.90	0.79	0.64
240×120	1	1.00	0.98	0.95	0.89	0.78

Table1 : Performance figures (efficiency) for the 2D module of Grid∗ on Ncube. A communication step is performed after 10 iterations. A system of 2 nonlinear Poisson equations is solved by SOR.

Intel iPSC2 System						
points per proc	number of processors					
	1	2	4	8	16	32
30×30	1	0.99	0.95	0.91	0.68	0.59
60×30	1	0.99	0.96	0.93	0.75	0.63
60×60	1	0.99	0.98	0.96	0.73	0.68
120×60	1	1.00	0.98	0.96	0.72	0.67
130×120	1	1.00	0.99	0.94	0.73	0.72
240×120	1	0.99	0.99	0.95	0.76	0.73

Table2 : Performance figures (efficiency) for the 2D module of Grid* on the Intel iPSC2. A communication step is performed after 1 iteration.

FPS 500 System						
points per proc	number of processors					
	1	12	28	36	56	84
240×120	1	0.99	0.99	0.76	0.74	0.97
240×120	1	0.99	0.99	0.97	1.00	0.97

Table3 : Performance figures (efficiency) for the 2D module of Grid∗ on the FPS 500 Matrix Processor. One communication step after 20 iterations. The first row depicts efficiency for a fixed number of 84 blocks, which results in poor load balancing for the 36 and 56 processor configuration. Efficiency values of the second row are for exactly 1 block per processor, leading to a nearly perfect load balancing [2].

7 Conclusions and Outlook

A 2D multiblock code was used as a testcase for several massively parallel systems. It has been shown that teh multiblock approach, which is somewhere between the completely structured and the completely unsructured approach, is ideally suited for a MIMD architecture. It is believed that this type of architecture is very suitable to handle the high communication load that results from a loadbalanced 3D multiblock Navier-Stokes code, and at the same time can be extended to provide several thousand powerful processors in order to achieve the compute power needed in a cost effective way. Guidelines have been given how to modify a sequential code to exhibit a large amount of inherent paralelism in order to alleviate the port to a massively parallel system. A large multiblock code can also be ported on a shared memory system, but it is doubtful whether thousands of processors could be supported. For a SIMD architecture good results can be achieved [12], but major modifications would be needed for the codes described in this paper.

8 Acknowledgement

The continuous support and encouragement of W. Berry, Propulsion and Aerothermodynamics Division, ESTEC is gratefully acknowledged. The authors are indebted to R.D. Williams, Caltech, for numerous helpful discussions. The help and contribution of staff from FPS, Intel, Ncube, and Thinking Machines is sincerely appreciated.

References

[1] Häuser, J. et al., 1990 : Parallel Computing in Aerospace Using Multi-Block Grids:Part I: Application to Grid Generation, submitted to Concurrency, Practice and Experience, 23pp., preprint available from the authors.

[2] Fps Computing, 1991: Application of Parallel Computing using Multi-Block Grids on the FPS 500 Series Matrix Pocessor.

[3] Fox,G. et al., 1988: Solving Problems on Concurrent Processors, Vol. 1, Prentice Hall, 592 pp.

[4] Haimes, R.D. et al., 1991 : Visual3 User's & Programmer's Manual, Rev. 1.25, MIT, 46pp.

[5] Häuser, J., Williams, R., 1991 : Strategies for Parallelizing a Navier-Stokes Code on the Intel Touchstone Machines, submitted to J. Num. Meth. in Fluids, 7pp., preprint available from the authors.

[6] Li, C.P., 1991 : Computational Methods for Shock Waves in Three-Dimensional Supersonic Flow, Computer Methods in Applied Mechanics and Engineering 87, pp.307-327.

[7] Williams, R.D., 1990 : Express : Portable Parallel Programming, pp. 347-352, Proc. Cray User Group, Austin Texas.

[8] Simon, H.D., Dagum, L., 1991 : Experience in Using SIMD and MIMD Parallelism for Computational Fluid Dynamics, NASA Ames Research Center, Report RNR-91-014.

[9] Bailey, D.H. 1991 : Experience with Parallel Computers at NASA Ames, NASA Ames Research Center, Report RNR-91-007.

[10] Barszcz, E., 1991 : One Year with an iPSC/860, NASA Ames Research Center, Report RNR-91-001.

[11] Häuser, J. et al., 1991 : **Grid$*$** : A General Multiblock Surface and Volume Grid Generation Tollbox in Numerical Grid Generation and Related Fields, eds A. Arcilla, J. Häuser,P.R. Eiseman, and J.F. Thompson, Elsevier North-Holland

[12] Egolf, T.A. et al., 1990 : Computational Fluid Dynamics on the Connection Machine at UTRC in Parallel Processing in Engineering Applications, ed. R.A. Adey, Southampton.

[13] Liu,Y., Shakib, F., Vinokur, M., 1990 : A Comparison of Internal Energy Calculation Methods for Diatomic Molecules, Phys. Fluids A2(10), pp. 1884-1902.

[14] Seidel, S.R. et al., 1991 : Concurrent Bidirectional Communication on the Intel iPSC/860 and iPSC/2 in The Sixth Distributed Memory Computing Conference Proceedings, Portland, pp. 283-286, IEEE Computer Society Press.

Title of publication: Proceedings of the Conference on "Parallel Computational Fluid Dynamics", Stuttgart, Germany, 10-12 June 1991

Editor(s): K.G. Reinsch et al.

Submission of your paper for publication is understood to imply that the article has not been published previously nor is it being considered for publication elsewhere. If parts of the material, e.g. text, figures or tables, have already been published elsewhere, permission for their reproduction must be obtained from the copyright owners and full acknowledgement made in the manuscript. The manuscript shall be submitted free of copyright charges.

Formal written transfer of copyright on each article from the author(s) to the publisher is needed. This transfer enables the publisher to provide for the widest possible dissemination of the article through activities such as: distribution of reprints; authorization of reprints, translation, or photocopying by others; production of microfilm editions; and authorization of indexing and abstracting services in print and data base formats. Without this transfer of copyright, such activities by the publisher and the corresponding spread of information are limited.

Please fill in:

Title of article for which copyright is being transferred: [illegible] Simulation [illegible] Parallel Systems

Authored by: Jochen M Hauser Horst D Simon

If the article has been written in the course of employment by a Government agency so that no copyright exists, please mark this box. [✓]

Otherwise, the copyright is hereby transferred to Elsevier Science Publishers B.V., effective if and when the article is accepted for publication.

The author(s) reserve(s) the following:

(1) All proprietary rights such as patent rights.
(2) The right to use all or part of this article in future works of their own, such as lectures, press releases, reviews or text books.
(3) The right of reproduction and limited distribution for own personal use or for company use for individual clients, provided that source and copyright are indicated and copies per se are not offered for sale.

In the case of republication of the whole article or parts thereof by a third party, written permission should be obtained from Elsevier Science Publishers B.V., to be signed by at least one of the authors (who agrees to inform the others, if any) or, in the case of a "work made for hire", by the employer.

This form must be completed and signed by the author(s), and received by the specified Editor's office, before the article can be accepted for publication. In the case of an article commissioned by another person or organization, or written as part of duties as an employee, an authorized representative of the commissioning organization or employer must sign instead of the author.

Space for signature of co-authors:

Signature [illegible]

Print Name Jochen M. Hauser

Title/Employer
(if not signed by author)

If the form is being signed by only one author on behalf of the others, the following additional statement must be accepted and signed by the author signing for the co-authors: "I represent and warrant that I am authorized to execute this transfer of copyright on behalf of all the authors of the article named above."

Date [illegible]

Signature [illegible]